So, in the last lecture, I showed you this technique to inject a real or a fake FileReader for this video service object. This approach works but it's not ideal in enterprise applications.

Because in a real world application this class might have a couple more dependencies you don't want to repeat this expression a few times and also make these parameters optional. It looks a little bit ugly.

That's why we refer to this approach as Poor man's dependency injection. In a real enterprise like application, you don't want to do this. You want to keep your code as simple as possible. That's why we use a dependency injection framework.

When you use a proper dependency injection framework in your application, you can simplify this constructor to something like this, like this. It's far simpler. So a dependency injection framework will take care of creating and initializing objects at run time. There are various dependency injection frameworks out there. We have name jack (?), we have structure map, spring.net, autofac, unity, and so on.

Almost all these frameworks follow the same principles. In a dependency injection framework, you have a container, this container is a registry of all your interfaces and their implementations. When your application starts, your dependency injection framework will automatically take care of creating object graphs based on the interfaces and types registered in the container. For example, in an ASP.NET dependency application, when you receive a request on the server, ASP.NET runtime. Should create an instance of a controller class. Your controller may have one or more dependencies. Let's say IFileReader. At this point dependency injection framework kicks in, it looks at the parameters of the contructor or controller, then it looks at this container or registry, and finds the concrete implementations for these referenced interfaces, instantiates them, and passes them to your controller. If those objects also have dependencies, your dependency injection framework (?) will take care of creating and initializing an object graph for you. Using a dependency injection framework is beyond the scope of this course and that's something that you need to research on your own, because the implementation varies from one framework to another, and also it's dependent on the kind of application you're building. For example, the set of code for using a dependency injection framework in an ASP.NET application is different from than let's say a Xamarin or WPF application.

It's even different amongst different versions of ASP.NET. .MPC. So, if you want my recommendation, I would say go with either Ninject, or Aurofac These are the two popular dependency injection frameworks out there. By the end you might be part of a team, and your team later may love a different dependency injection framework, that's perfectly fine. Don't get hung up on the tooling. So, pick one of these dependency injection frameworks read their documentation about what you need to do to use them in your applications.